SCIENTIFIC COMPUTING

Dr. Johnson

School of Mathematics

Semester 1 2012
1 Course Objectives

- Structure of the course
- Object-Orientated Programming
OUTLINE

1. COURSE OBJECTIVES
   - Structure of the course
   - Object-Orientated Programming

2. THE BASICS
   - What is a computer?
   - Syntax
OUTLINE

1 Course Objectives
   - Structure of the course
   - Object-Orientated Programming

2 The Basics
   - What is a computer?
   - Syntax

3 Understanding Data
   - Handling Data and Variables
   - Namespaces
   - Simple Input and Output
1. **Course Objectives**
   - Structure of the course
   - Object-Orientated Programming

2. **The Basics**
   - What is a computer?
   - Syntax

3. **Understanding Data**
   - Handling Data and Variables
   - Namespaces
   - Simple Input and Output

4. **Summary**
1 COURSE OBJECTIVES
   • Structure of the course
   • Object-Orientated Programming

2 THE BASICS
   • What is a computer?
   • Syntax

3 UNDERSTANDING DATA
   • Handling Data and Variables
   • Namespaces
   • Simple Input and Output

4 SUMMARY
5 weeks of lectures/Tutorials consisting of

- The Basics:
  - Data and variables;
  - Input and output;
  - Flow Control;
  - Functions;
  - Dynamic Allocation and Pointers;
  - Using Standard Libraries.
Understanding Basic C++

5 weeks of lectures/Tutorials consisting of

- **The Basics:**
  - Data and variables;
  - Input and output;
  - Flow Control;
  - Functions;
  - Dynamic Allocation and Pointers;
  - Using Standard Libraries.

- **Object-Orientated programming:**
  - Constructor, deconstructors, members, public and private;
  - Operator overloading;
  - Inheritance, virtual members and runtime polymorphism.
By the end of the course you should be able to:

- Understand basic concepts of writing a program
- Convert mathematical algorithms into C++ code
By the end of the course you should be able to:

- Understand basic concepts of writing a program
- Convert mathematical algorithms into C++ code
- Interpret the output from a program, producing figures and/or tables to show results
- Use the concepts of Object-Orientated programming to improve your programs
By the end of the course you should be able to:

- Understand basic concepts of writing a program
- Convert mathematical algorithms into C++ code
- Interpret the output from a program, producing figures and/or tables to show results
- Use the concepts of Object-Orientated programming to improve your programs
- Write a technical report on a numerical method
1. **Course Objectives**
   - Structure of the course
   - Object-Orientated Programming

2. **The Basics**
   - What is a computer?
   - Syntax

3. **Understanding Data**
   - Handling Data and Variables
   - Namespaces
   - Simple Input and Output

4. **Summary**
Why Objects?

- A structured language can *hide* information from the rest of the program.
- Structuring code and data allows
  - easy upgrades
  - many programmers to work on a large project
**Why Objects?**

- A structured language can *hide* information from the rest of the program.
- Structuring code and data allows
  - easy upgrades
  - many programmers to work on a large project
- Object-oriented programming imposes a high level of structure
- Problems are broken down into subproblems, and then into self-contained units called objects
Why Objects?

- A structured language can *hide* information from the rest of the program.
- Structuring code and data allows
  - easy upgrades
  - many programmers to work on a large project
- Object-oriented programming imposes a high level of structure
- Problems are broken down into subproblems, and then into self-contained units called objects
- Common traits of object-oriented languages are:
  - encapsulation
  - polymorphism
  - inheritance
 USING OBJECTS

1. **Encapsulation:**
   functions and data inside an object have restricted access.
**Using Objects**

1. **Encapsulation:**
   functions and data inside an object have restricted access.

2. **Polymorphism:**
   represents the concept of “one interface, multiple method”. The same interface can be used to do different things for different objects: i.e. define + to add numbers, but perform string concatenation on characters and strings, ‘a’ + ‘b’ = ‘ab’.
Using Objects

1. **Encapsulation:**
   functions and data inside an object have restricted access.

2. **Polymorphism:**
   represents the concept of “one interface, multiple method”. The same interface can be used to do different things for different objects: i.e. define + to add numbers, but perform string concatenation on characters and strings, ‘a’ + ‘b’ = ‘ab’.

3. **Inheritance:**
   allows one object to acquire the properties of another. An example would be to define a generic object “car” that has a steering wheel, four wheels and an engine. The new object “sports car” inherits all these properties and adds a sun roof, go-faster stripes and a huge stereo.
Topics:
- Computers and Programs;
- Syntax and Structure of a Program;
- Data and Variables;

Aims:
- Understand the idea of programming a computer;
- Write a simple program to input and output data.
1 Course Objectives
- Structure of the course
- Object-Orientated Programming

2 The Basics
- What is a computer?
- Syntax

3 Understanding Data
- Handling Data and Variables
- Namespaces
- Simple Input and Output

4 Summary
**Course Objectives**
**The Basics**
**Understanding Data**
**Summary**

**What is a computer?**

**Syntax**

---

**An Idealized Computer**

- **CPU - Central Processing Unit**
- **CPU and memory work together**

[Diagram of an Idealized Computer]

- Input
- CPU
- Memory
- Output
- File Store
**Course Objectives**

**The Basics**

**Understanding Data**

**Syntax**

**What is a computer?**

**AN IDEALIZED COMPUTER**

- CPU - Central Processing Unit
- CPU and memory work together
- Input may be from keyboard, mouse or a file
- Output may be to screen or a file
What is a computer?

A program is a sequence of instructions that enable a computer to complete a task. Computers can remember programs. Originally, programs had to be written in machine code, which is low-level. Now, programs are written in high-level code, and the computer generates the machine code for us. C/C++ are best described as mid-level languages. Higher-level languages include Fortran, Matlab, and others.
**What is a program?**

A program is sequence of instructions to enable a computer to complete a task.

Computers can remember programs. Originally programs had to be written in machine code - low-level. Now we write programs in code - and let the computer write the machine code for us.

C/C++ are best described as mid-level languages. Higher-level languages include Fortran, Matlab, and others.
**What is a program?**

A program is a sequence of instructions to enable a computer to complete a task.

- Computers can remember programs.
- Originally programs had to be written in machine code - *low-level*.
What is a program?

A program is sequence of instructions to enable a computer to complete a task.

- Computers can remember programs
- Originally programs had to be written in machine code - low-level
- Now we write programs in code
What is a program?

A program is a sequence of instructions to enable a computer to complete a task.

- Computers can remember programs.
- Originally programs had to be written in machine code - low-level.
- Now we write programs in code - *and let the computer write the machine code for us!*
- C/C++ are best described as mid-level languages.
- Higher-level languages include Fortran, Matlab, and others.
COURSE OBJECTIVES
- Structure of the course
- Object-Orientated Programming

THE BASICS
- What is a computer?
- Syntax

UNDERSTANDING DATA
- Handling Data and Variables
- Namespaces
- Simple Input and Output

SUMMARY
The key elements of C/C++ syntax are:
- Semicolon used to mark end of statements
- Case is important
- Totally free form, lines and names can be as long as you like!
- Comments take the form /* C style comment */ or // C++ style comment
- Code blocks are surrounded by braces {}
STRUCTURING YOUR CODE

- Include libraries and methods
- Declare data types and variable names
- Input values for variables
**STRUCTURING YOUR CODE**

<table>
<thead>
<tr>
<th>Libraries/Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
</tr>
<tr>
<td>Declare</td>
</tr>
<tr>
<td>Input</td>
</tr>
<tr>
<td>Calculate</td>
</tr>
<tr>
<td>Output</td>
</tr>
</tbody>
</table>

- Include libraries and methods
- Declare data types and variable names
- Input values for variables
- Required calculations are carried out in sequence
- Output to the screen or a file
A Very Simple C++ Code

- The following is a C++ program.

```cpp
main()
{
}
```

- There are no commands to execute, so the program will not do anything.
The following is a C++ program.

```c++
main()
{
}
```

There are no commands to execute, so the program will not do anything.

Please see notes and web tutorials on how to compile and run your programs!
A C++ code is saved as an ASCII text file and can written in any editor.

However, in this course we will encourage the use of an Integrated Development Environment (IDE).

Three recommended pieces of software are:

- Visual Studio (Windows only)
- Netbeans (Windows/Linux/Mac)
- Kdevelop (Linux KDE)

For our purposes in this course, it will not matter on which platform you write the code.
Unlike higher level programming languages, there are almost no intrinsic functions in C++

This includes the ability to print to screen.

We can include standard libraries for:
- Input/Output
- Advanced Storage
- Strings
- Mathematical functions

The syntax for including libraries is:

```
#include <library_name>
```

Include statements must appear before any other statements.
A simple example of the standard input/output library:

```cpp
// Divide two integers
#include <iostream>
using namespace std;

main(){
    int i, j;
    cout « 'Input two integers' « endl;
    cin » i » j;
    cout « ' i/j = ' « i/j « endl;
}
```
1 Course Objectives
   - Structure of the course
   - Object-Orientated Programming

2 The Basics
   - What is a computer?
   - Syntax

3 Understanding Data
   - Handling Data and Variables
   - Namespaces
   - Simple Input and Output

4 Summary
Computers are not humans!

Computers can only follow the instructions they are given. If they are not doing what you want it is because you haven’t told them explicitly what to do!
Computers are not humans!

Computers can only follow the instructions they are given. If they are not doing what you want it is because you haven’t told them explicitly what to do!

- One of the most common mistakes when you programming is caused by swapping between data types
- Program may not work quite how we want...
Computers are not humans!

Computers can only follow the instructions they are given. If they are not doing what you want it is because you haven’t told them explicitly what to do!

- One of the most common mistakes when you programming is caused by swapping between data types
- Program may not work quite how we want...
- So we must understand how the computer uses data.
There are six basic data types in C++:

- `char` – a character
- `(short/long) int` – integers with different sizes
- `float` – single precision real number
- `(long) double` – double (or higher) precision number
- `bool` – true or false binary number
- `void` – this is used when a function doesn’t return a value

The precision of numbers will effect the maximum and minimum values they can take

and also how big errors are.
Declaring Variables

- A variable is a named location in memory, used to store data.
- We may declare variables anywhere in the code.
- Variables will be localised to the block in which they are declared.
Declaring Variables

- A variable is a named location in memory, used to store data.
- We may declare variables anywhere in the code.
- Variables will be localised to the block in which they are declared.
- What is the output from the following?

```cpp
int i=0;
cout << " i= " << i << endl;
{
  int i=10;
  cout << " i= " << i << endl;
}
cout << " i= " << i << endl;
```
**Using Variables**

- We can assign a variable a value using `='

```
data_1 = 10. + 21.5
```

- The data types on both sides of `=' must be compatible
Using Variables

- We can assign a variable a value using `='

```plaintext
data_1 = 10. + 21.5
```

- The data types on both sides of `=' must be compatible
- We can add/subtract, multiply/divide standard data types

```plaintext
double a,b,c;
b = 5. ; c = 4.1;
a = 10. * b + c;
```
Other Operators

- We call \( +, -, *, / \) operators.
- We shall see later that they may be overloaded.
- Other operators are:
  - \( a \% b \) :: returns \( a \) modulus \( b \)
  - \( a++ \) :: increment \( a \) by 1
  - \( a-- \) :: decrement \( a \) by 1
  - \( a+=b \) :: set \( a \) equal to \( a \) plus \( b \)
  - \( a*=b \) :: set \( a \) equal to \( a \) times \( b \)
OUTLINE

1. COURSE OBJECTIVES
   - Structure of the course
   - Object-Orientated Programming

2. THE BASICS
   - What is a computer?
   - Syntax

3. UNDERSTANDING DATA
   - Handling Data and Variables
   - Namespaces
   - Simple Input and Output

4. SUMMARY
WHAT IS IN A NAME?

- Each variable requires a unique name (in the same scope)
- In large projects this becomes very difficult
- A namespace is like adding a *surname* to a variable
- The prefix `std` means functions from the standard library
- We can use the statement:

```cpp
using namespace std;
```

to assume an undeclared function, data type or variable is in the standard library.
Outline

1. Course Objectives
   - Structure of the course
   - Object-Orientated Programming

2. The Basics
   - What is a computer?
   - Syntax

3. Understanding Data
   - Handling Data and Variables
   - Namespaces
   - Simple Input and Output

4. Summary
We use *stream* variables to access the screen, keyboard and files.

- They allow us to associate a name with a physical output.
- We need to include stream libraries at the top of the program.

```cpp
#include<iostream>
using namespace std
main()
{ 
    int i
    cout << " Enter a number. " << endl;
    cin >> i; //read in a number
    cout << " Your number is " << i << endl;
}
**Simple Input and Output**

- `cout` is the standard screen variable, and `cin` the standard keyboard variable.
- To pass data to and from the *stream* we use the « and » operators.
**Simple Input and Output**

- `cout` is the standard screen variable, and `cin` the standard keyboard variable.
- To pass data to and from the *stream* we use the « and » operators.
- « data is passed right to left, in the example the string is passed to `cout`.
- » data is passed left to right, in the example the integer is passed from `cin` to `i`.

Use `endl` to finish a line.
Simple Input and Output

- `cout` is the standard screen variable, and `cin` the standard keyboard variable.
- To pass data to and from the *stream* we use the « and » operators.
- « data is passed right to left, in the example the string is passed to `cout`.
- » data is passed left to right, in the example the integer is passed from `cin` to `i`.
- Multiple bits of data can be passed to the stream by stringing them together in the same command.
- Use `endl` to finish a line.
To read and write to files we must include the `fstream` library.

Input streams have type `ifstream`, and output streams `ofstream`.

```cpp
ifstream file_input; // an input file stream
ofstream file_output; // an output file stream
```

`ifstream` and `ofstream` have intrinsic functions to open and close files.

We can also check if the file is open with the `is_open()` function.

```cpp
file_input.open("input.in"); // open file
```
Topics:
- Computers and Programs;
- Syntax of C++;
- Data and Variables;
- Input and Output.

Aims:
- Understand the idea of programming a computer;
- Write a simple program to input and output data.